IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Patent Application of:
David Boreham et al.

Conf. No.: 6566

Application No.: 09/867,595

Art Unit: 2167

Filed: May 29, 2001

Examiner: K. S. Lu

For: METHOD AND SYSTEM FOR DETERMINING A DIRECTORY ENTRY CLASS OF SERVICE BASED ON THE VALUE OF A SPECIFIER.

DECLARATION OF KIERAN CROSS ROWLEY

1. I, Kieran Cross Rowley, was a technical writer at Netscape Communications Corp. (which was part of the iPlanet alliance between Sun Microsystems, Inc. and Netscape Communications Corp.) and wrote the Sun Microsystems, Inc. "iPlanet Directory Server Administrator's Guide, Version 5.0," published April 2001 (hereafter "iPlanet").

2. I, Kieran Cross Rowley, received all the technical content of iPlanet from David W. Boreham and Peter Rowley.

3. I, Kieran Cross Rowley, used the document written by David W. Boreham and Peter Rowley entitled "Netscape Directory Server 5.0 Class of Service Facility" to write iPlanet. The aforementioned document is attached as Appendix A.

4. I, Kieran Cross Rowley, had numerous meetings with David W. Boreham and Peter Rowley to discuss and review the technical content of the Class of Service Facility described in iPlanet.

I, Kieran Cross Rowley, hereby declare that all statements made herein of my own knowledge are true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Signed this day 12 July, of July 2005. Kieran Cross Rowley
Kieran Cross Rowley with permission/bcm

Appendix A
Directory Service 5.0 Roles Overview

Directory Server 5.0 Roles Overview

David Boreham August 9, 1999

What is a Role ?

DS5.0 Roles embody a new comprehensive entry grouping mechanism, similar to the existing Directory Server *group* concept. Roles address many of the known shortcomings present in static and dynamic groups. Roles unify all entry grouping functions within one mechanism that is both easy to use and understand. Terminology primer: each role has *members*, which are the entries said to *possess* the role. In order to achieve reasonable implementation efficiency and to support certain important management functions, two different varieties of roles are envisaged. These are termed the *simple* and *complex* role types.

How are Roles Defined?

Each and every role is defined by its own role definition entry. A role is uniquely identified by the distinguished name of its defining entry. Role definition entries are LDAP *subentries* and so inherit the subentry mechanism for scope of application to a DIT subtree. LDAP Subentries are defined in

That each and every role is defined by an entry stored in the DIT is of fundamental importance. Any client with appropriate access rights can discover, identify and examine any role definition. Any client with appropriate access rights can add a new role definition and modify existing role definitions. Role definitions may be replicated in a distributed environment.

Role Operations

Given arbitrary roles and entries, the following operations are useful to clients:

1. Enumerate the members of a role. (Answer the question "which entries have this role ?"). It's also useful to be able to resolve this query in a reasonable time (significantly less than the time to find the members by brute force examination of all the entries). It's also useful to be able to retrieve the entries in a paged or browsing fashion, using the existing LDAP VLV mechanism.
2. Determine whether a given entry possesses a particular role. It's useful to be able to do this more efficiently than by determining all the roles possessed by the entry and then checking whether the target role is among that set of roles.
3. Enumerate all the roles possessed by a given entry.
4. Assign a particular role to a given entry.
5. Revoke a particular role from a given entry.

Tell us about the two Kinds of Role

Certainly. Imagine that all roles were treated similarly. A role definition could be as complex as required or desired. An example of a complex role would be a nested role whose component roles included filters using custom matching rules and attribute syntax. Now consider the set of operations we desire to support for roles, listed above:

1. Membership enumeration: to enumerate the members, a client would need to separately enumerate the component roles, by submitting search operations to the server. The results would need to be aggregated by the client, eliminating any duplicates. While this is possible, it's onerous.
2. Determine whether a given entry possesses this role: this is really hard for a client to do. A brute force method would be terribly inefficient. Some associative indexing scheme could potentially help but seems hard to implement well.
3. Enumerate all the roles possessed by a given entry: this is harder than (2).
4. Assign this role to a given entry: This is really hard, requiring semantic knowledge of the role definition.
5. Just as hard as (4).

So, we have asserted that for a complex role such as the example used above, it is hard or very hard to implement any of the five desirable role operations. Therefore we define a role type with deliberately limited flexibility, called a *simple role*, which *does* allow all five of the desirable role operations to be implemented easily and efficiently. Any role that is not a simple role is by definition a *complex role*.

A simple role has the following properties:

1. It's easy and efficient to enumerate its membership. This enumeration can be done using VLV controls, allowing better scaling and user experience.
2. It's easy and efficient to test an entry for possession of a particular role.
3. It's easy and efficient to enumerate the simple roles a particular entry possesses.
4. It's obvious by inspection of the role definition how to assign a simple role to a given entry.
5. It's obvious by inspection of the role definition how to revoke a simple role from a given entry.

Roles which are not simple are deemed complex, with the implication that one or more (perhaps all) of the five operations are hard or very hard. Clients and management agents can easily determine whether a given role is of the simple or complex kind, allowing UI to act appropriately. Various server features which use roles will not be supported, or supported in a limited or inefficient way, when the target role is complex. Internal to the server, it is also possible to deliver notification to subsystems when entries move in and out of simple group membership. This can be used to keep metadata cached for performance reasons coherent with the DIT contents.

Roles Compared to Legacy Grouping Mechanisms

	DS1.x-4.x Static Group	DS4.x Dynamic Group	Messaging Server Dynamic Group	Role
Definition in the DIT	Yes	No	Yes	Yes
Means something to the server	Yes	Yes	No	Yes
Single unified grouping mechanism	No	No	No	Yes
Supported by Console UI	Yes	No ?	No	Yes
Usable for DS Access Control	Yes	Yes	No	Yes

Usable for Application Logic	Yes	No	Yes	Yes
Easy and efficient membership enumeration	Yes	No	Yes	Yes
Can use VLV to enumerate membership	No	No	Yes	Yes
Server performs membership check for clients	No	No	No	Yes
Scales to large membership	No	Yes	Yes	Yes
Supports generated attribute values and other cool stuff	No	No	No	Yes

Role Definitions, Take Two

Having covered the difference between simple and complex roles, let's return to the details of role definition schema:

There are two "abstract" object classes to distinguish between simple and complex roles: nsSimpleRole and nsComplexRole.

Currently there is only one variety of simple role: the *managed* role with object class nsManagedRole. Membership of a managed role is conferred upon an entry by adding the role's DN to the entry's nsRoleDN attrbitute. Example of a managed role definition:

```
dn: cn=managerrole, dc=dom, dc=ain
objectclass: top
objectclass: LDAPsubentry
objectclass: nsRole
objectclass: nsSimpleRole
objectclass: nsManagedRole
cn: managgerrole
description: Manager Role within the Company
```

Example of an entry with this role:

```
dn: uid=pointyhair, ou=people, dc=dom, dc=ain
objectclass: top
objectclass: person
objectclass: inetorgperson
uid: pointyhair
gn: pointy
```

```
sn: hair
nsRoleDN: cn=managerrole, dc=dom, dc=ain
description: Pointy Haired Manager
```

There are currently three kinds of complex role: filtered, nested and enumerated roles, with corresponding object classes nsFilteredRole, nsNestedRole and nsEnumeratedRole.

A filtered role specifies an LDAP filter of arbitrary complexity. Entries matching the filter and within the sub tree specification are said to possess the role. Here's an example of a filtered role definition:

```
dn: cn= building12people, dc=dom, dc=ain
objectclass: top
objectclass: LDAPsubentry
objectclass: nsRole
objectclass: nsComplexRole
objectclass: nsFilteredRole
cn: building12people
nsRoleFilter: building=12
description: People who work in building 12
```

A *nested* role specifies containment of one or more roles of any type. Here's an example of a nested role definition:

```
dn: cn=megamanagerrole, dc=dom, dc=ain
objectclass: top
objectclass: LDAPsubentry
objectclass: nsRole
objectclass: nsComplexRole
objectclass: nsNestedRole
cn: megamanaggerrole
nsRoleDN: cn=managerrole, dc=dom, dc=ain
nsRoleDN: cn=networkmanager, dc=dom, dc=ain
description: Super Manager Role within the Company
```

An enumerated role is the static group reborn, but with one important difference. No nesting is allowed within an enumerated role. This permits a valuable optimization not possible with the DS1/3/4.x static groups. In addition, it is illegal to add a member to an enumerated group where that entry lies outside the subtree specification. [Note: I'm not sure we can implement enumerated roles efficiently. Beware that DS5.0 may not implement them.]

```
dn: cn=peonrole, dc=dom, dc=ain
objectclass: top
objectclass: LDAPsubentry
objectclass: nsRole
objectclass: nsComplexRole
objectclass: nsEnumeratedRole
objectclass: groupofuniquenames
cn: peonrole
groupmember: uid=dboreham, ou=people, dc=dom, dc=ain
```

```
groupmember: uid=merrells, ou=people, dc=dom, dc=ain
groupmember: uid=claire, ou=people, dc=dom, dc=ain
description: Lowest of the Low Role within the Company
```

Role Operations on the Different Role Types

A brief description of how to implement each of the five fundamental role operations within each of the four role types described above (were possible):

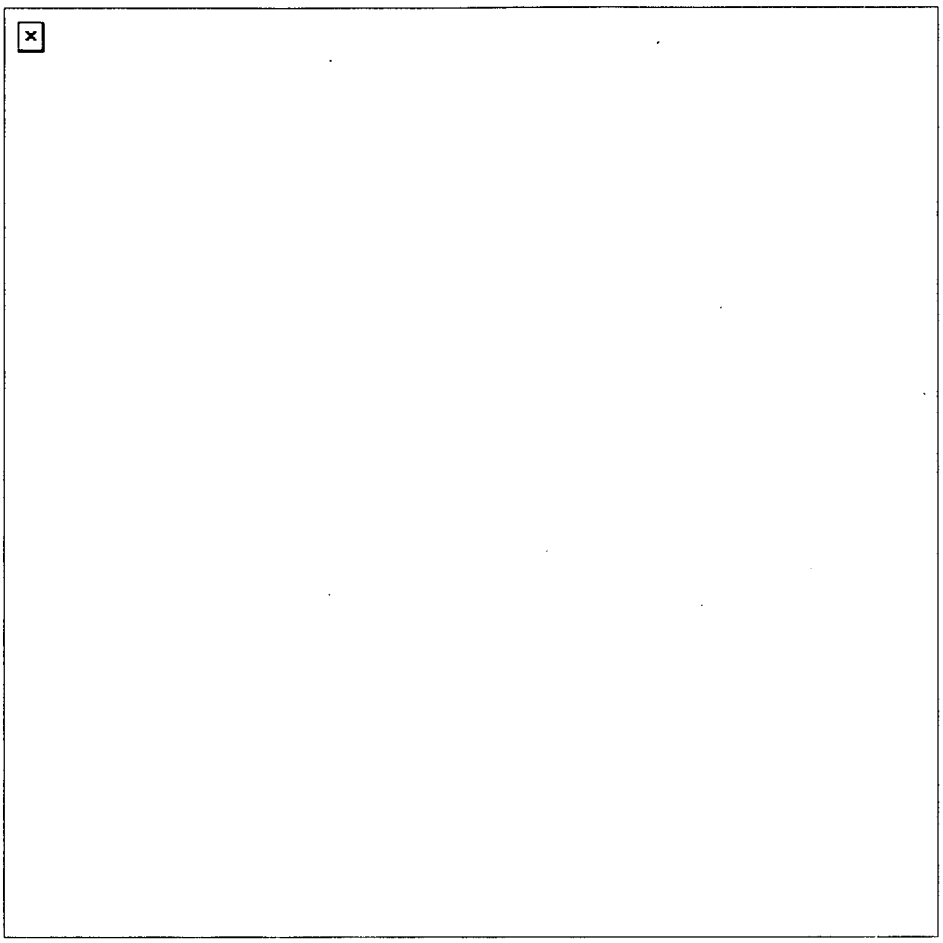
	Managed Role	**Filtered Role**	**Nested Role**	**Enu**
Membership enumeration	LDAP search with filter nsRoleDN=<role's dn> (guaranteed to be indexed and to support VLV).	LDAP subtree search with baseDN from the subtreeSpecification and filter from the nsRoleFilter attribute. (VLV indexing optional)	No generic mechanism, probably hard.	No mec hard
Membership Test	LDAP compare against nsRoleDN attribute	LDAP compare against nsRole attribute	LDAP compare against nsRole attribute	LDA agai attri com agai grou attib
Role Enumeration	Read nsRoleDN attribute	Read nsRole attribute	Read nsRole attribute	Rea
Assign Role	LDAP modify, add role's DN to nsRoleDN attribute	Not meaningful	Not meaningful	Add the men
Revoke Role	LDAP modify, remove role's DN from nsRoleDN attribute.	Not meaningful	Not meaningful	Rem from men

	Any role	**Simple Role**	**Complex Role**
Membership enumeration	No generic mechanism.	LDAP search with filter	No generic mechanism, probably

		nsRoleDN=<role's dn>	hard.
Membership Test	LDAP compare operation against nsRole attribute	LDAP compare against nsRoleDN attribute	LDAP compare against nsRole attribute
Role Enumeration	Read nsRole attribute	Read nsRoleDN attribute	Read nsRole attribute
Assign Role	Not generically defined	LDAP modify, add role's DN to nsRoleDN attribute	Not generally meaningful
Revoke Role	Not generically defined	LDAP modify, remove role's DN from nsRoleDN attribute.	Not generally meaningful

Role-Defined Attribute Values

Covered in detail elsewhere, but an important part of the role story, role-defined attribute values (formerly known as *Class of Service,* also known as *Computed Attributes*) significantly extend the value of roles. Using definitions stored in the DIT, the server is able to generate entry attribute values on-the-fly, based on the roles possessed by the entry. Not only is this useful to applications, for example in the case of the canonical mailbox quota example, but it's very useful within the server. For example, role-defined attributes may be used to control the server's *lookthrough limit* and *size limit* on an entry by entry basis.



For DS5.0, the existing Class of Service mechanism is extended to use an entry's role by simply allowing "nsRole" and "nsRoleDN" to be specified as the cos specifier attribute in a cosDefinition entry. DS4.1 Class of Service restricted the defining attribute to be single-valued. For DS5.0 role-defined attributes, this isn't possible (because nsRole and nsRoleDN can be multivalued). Consequently it becomes possible to define COS schemes where the choice of template entry is ambiguous. In order to resolve the ambiguity, the template entries may include a priority, in the cosPriority attribute, from the new *cosTemplate* object class.

```
dn: cn=messaging_server_cos, cn=cosdefinitions, o=NetscapeRoot
objectclass: top
objectclass: LDAPsubentry
objectclass: cosDefinition
cosTemplateDn: cn=LocalConfig, cn=MailServerCOS
cosSpecifier: nsRoleDN
cosAttribute: mailboxquota override
cosAttribute: maysendexternalmail default
```

Corresponding example template entries:

```
dn: cn="cn=guestnrole, dc=dom, dc=ain", cn=MailServerCOS, cn=LocalConfig,
o=NetscapeRoot
objectclass: top
```

```
objectclass: cosTemplate
objectclass: mailserveruser
cosPriority: 0
mailboxquota: 10000000
maysendexternalmail: TRUE

dn: cn="cn=userrole, dc=dom, dc=ain", cn=MailServerCOS, cn=LocalConfig,
o=NetscapeRoot
objectclass: top
objectclass: cosTemplate
objectclass: mailserveruser
cosPriority: 1
mailboxquota: 1000000
maysendexternalmail: FALSE
```

Role-Based Access Control

Access control rules can be defined such that the roles posessed by an entry determine its access rights. The aci syntax will be extended to support a "role" property Details to be found in other documents [which ones ?].

Access Control to Roles

Because management of managed roles requires access rights to modify the target entries, it becomes important to control access appropriately to the nsRoleDN attribute. Access control rules can be defined which restrict modification access rights to just the nsRoleDN attribute. Furthermore, access control rules can be defined such that a particular client is restricted to adding or removing a specific set of *values*. This means that a particular client will be able to, say, assign and remove access to the accounting application (which is done by means of a managed role), but the same client can not promote users to the system administrator managed role.